design**feature** *By Jan Axelson, Lakeview Research*

**IS UPDATING TO 2.0 WORTH IT? USB-PERIPHERAL DESIGN-
ERS NEED TO KNOW WHAT CHANGES IN THE SPEC MEAN
AND HOW THE INTERFACE HANDLES THE CHALLENGE OF
SUPPORTING HIGHER DATA RATES WHILE STILL ALLOWING
LOW- AND FULL-SPEED DEVICES ON THE BUS.**

Inside USB 2.0: what the new spec means for developers

JUST ABOUT THE TIME developers were starting to gain experience with USB, along came version 2.0. The new specification, released in April 2000, describes a high-speed option of 480 Mbps, 40 times faster than USB 1.1. A major goal of USB is to replace most of the traditional ports on a PC with one versatile and user-friendly interface.

A consortium of companies worked together to develop the USB specification, releasing version 1.0 in 1996 and updating it to 1.1 in 1998. (For a more detailed description of USB 1.x, see **Reference 1**). Microsoft first included USB support in OEM releases of Windows 95 and has added drivers, bug fixes, and other improvements to each new edition of Windows. Just about every PC from the past few years has at least one USB port. Apple's Macintosh also supports USB, as well as USB hardware and firmware. The specification names Compaq, Hewlett Packard, Intel, Lucent, Microsoft, NEC, and Philips as its author/sponsors. You can download the specification for free from the USB Implementers Forum (www.usb.org).

Compared with the interfaces it replaces, USB is more flexible and thus more internally complicated. It's a shared serial bus that places most of the interface intelligence in the host computer, thus enabling less complex and less expensive peripherals. Although its creators designed USB as a desktop bus for standard peripherals, it is also an option for just about any device that previously would have used an RS-232 or parallel port.

The major change in the new spec is an increase in the maximum data rate. USB 1.x supported two bus speeds: full speed at 12 Mbps and low speed at 1.5 Mbps. Low-speed data rates target cost-sensitive peripherals, as well as mice and other devices that require flexible cables (without bulky shields). USB 2.0

runs at 480 Mbps—a jump that opens the interface to new applications and ensures that it won't be obsolete for a while.

Of course, bus speed is different from the rate at which a device can actually transfer data. The data-transfer rate depends on how busy the bus is, as well as which of the spec's four transfer types is in use. In the best case, on an otherwise idle bus, a high-speed bulk transfer can move data at faster than 53 Mbytes/sec, using close to 90% of the bus's bandwidth for data.

USB's high-speed competitor is IEEE-1394, also known as Firewire. IEEE-1394 has a bus speed of 400 Mbps, and IEEE-1394b proposes to increase this rate

Figure 1
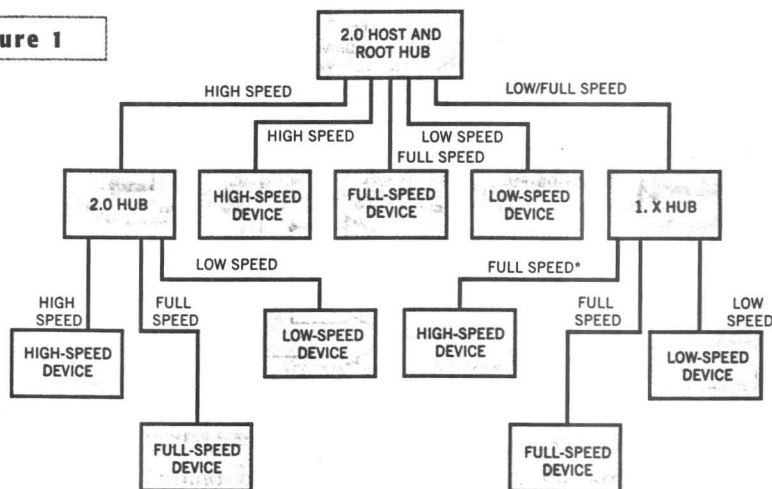
* FULL-SPEED ENUMERATION IS REQUIRED.
ADDITIONAL FULL-SPEED FUNCTIONS ARE OPTIONAL.

A USB 2.0 bus uses the high-speed bus rate whenever possible, switching to the low- or full-speed rate when necessary.

to 3.2 Gbps. Note that the two buses have different purposes, although some peripherals could use either device. With USB, the host initiates every transfer, and every transfer has one destination. With IEEE-1394, peripherals can communicate directly with each other, and a transfer can have multiple destinations. IEEE-1394 devices require more intelligence to manage their communications, so their peripheral controllers are more complex and expensive.

WHAT ABOUT THE USERS?

USB 2.0 makes it as easy as possible for end users to employ the high-speed data rates. However, users will need to install 2.0 host-controller hardware in their host computers and possibly upgrade their operating system as well.

In 2001, Intel will introduce Pentium 4 motherboards and a chip set with integrated USB 2.0 support. Systems without motherboard support will be able to add PCI cards with 2.0 controllers. Windows 2000 will support the controllers, and upgrades for Windows 98 SE and Windows Me will probably become available. To use the high-speed data rate, any hubs between the host and the device

must also be capable of running at the same rate. The high-speed cables are the same shielded, twisted pairs specified for devices with full-speed interfaces.

When the hardware is in place, users can proceed as usual. In most cases, they don't have to know or care about the speed of their devices or host controller. For example, when a high-speed device plugs into a hub that supports only low- and full-speed devices, the device must respond at full speed to the standard enumeration requests that identify the device. A high-speed device is required only to enumerate at full speed, but in most cases, it makes sense to make a product fully functional at both speeds. Doing so requires little extra development time or cost and increases the market for the device while 1.x controllers are still common.

REQUIREMENTS FOR A 2.0 DEVICE

Every USB peripheral requires a controller chip and firmware to manage its communications on the bus. The controller's hardware does most of the work of supporting high-speed data rates, including negotiating the switch from full speed to high speed when possible.

Vendors of 1.x chips, including Cypress, Lucent, and NetChip, currently offer or are developing high-speed peripheral controllers. For high-speed designs based on ASICs or FPGAs, Philips Semiconductor's ISP1501 transceiver can provide the hardware interface.

Intel has a peripheral-development kit that contains a 2.0 host controller on a PCI expansion card, a software stack for Windows 2000, and a single-step/debugging tool. The kit is available to members of the USB Implementers Forum. For testing, vendors, such as Catalyst Enterprises, Computer Access Technology Corporation, Crescent Heart Software, and Data Transit are either offering or developing 2.0 protocol analyzers.

Every USB device stores a series of descriptor tables that the host computer requests during enumeration to learn about the device's capabilities. The device descriptor includes a field to indicate the version of the spec that the device complies with. To support new abilities in 2.0, the spec also defines a few new fields in the descriptors and expanded options for some of the standard requests.

A host computer that supports USB 2.0 must have host-controller hardware

Figure 2

FULL/HIGH/LOW-SPEED DEVICE TRANSCEIVER
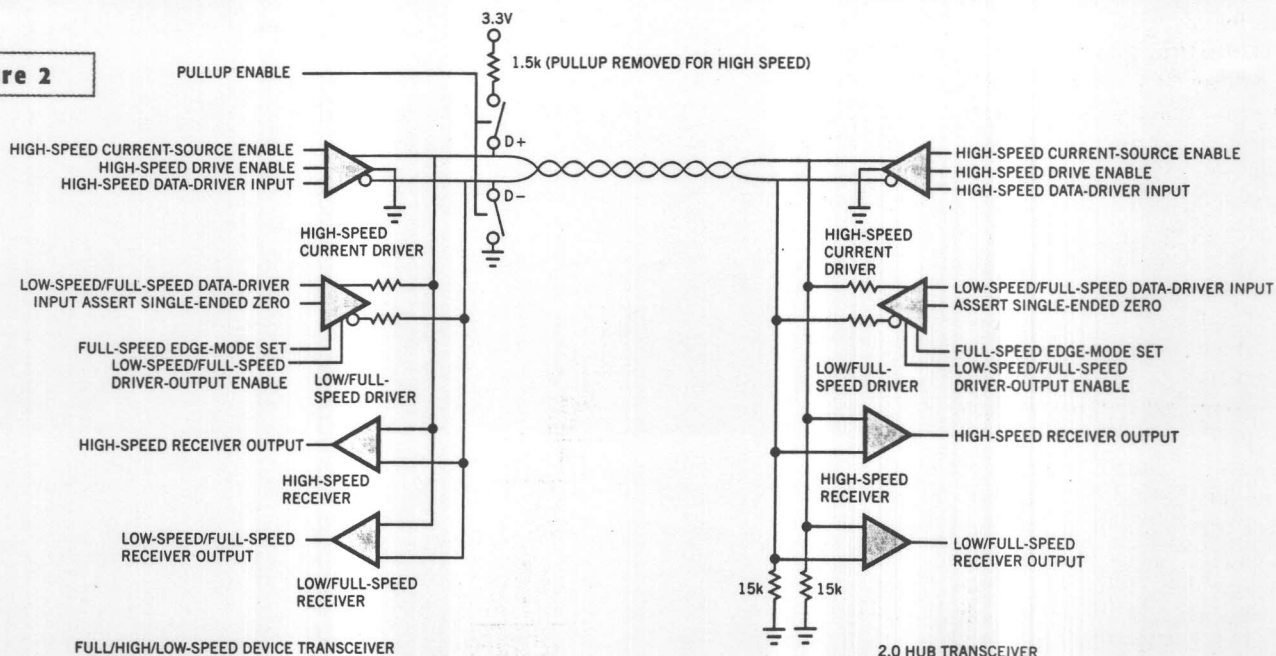
2.0 HUB TRANSCEIVER

NOTES:
WHEN A HIGH-SPEED DRIVER IS ACTIVE, THE FULL-SPEED DRIVERS ASSERT SINGLE-ENDED ZEROS, PROVIDING 45Ω TERMINATION.
NOT SHOWN: TRANSMISSION-ENVELOPE DETECTORS, DISCONNECTION-ENVELOPE DETECTORS, AND SINGLE-ENDED RECEIVERS.

A transceiver that supports the high-speed bus rate must also support the full-speed rate. During high-speed transmissions, the full-speed drivers and their series resistors terminate the line.

and software drivers (typically provided with the operating system) to handle low-level USB communications. Each device must also have a class or device driver to manage communication between the low-level drivers and applications that access the device. Windows and other operating systems include some class drivers. A device that can't use the operating system's built-in drivers must provide its own driver.

Application programmers and device-driver writers will be happy to hear that they don't have to worry about devising support for high-speed devices. Speed increases and other changes to the interface are all handled at a lower level. Applications and device drivers will require no changes to support faster data rates.

SUPPORTING THREE SPEEDS

High-speed USB 2.0 buses allow the use of low- and full-speed devices while transferring data at high speed whenever possible. On a full-speed bus, the host controller divides the bus time into 1-msec frames. Each frame begins with an SOF (start-of-frame) packet that devices use as a timing reference. Within each frame, the host can schedule multiple transactions to multiple destinations. Each transaction includes an endpoint address that identifies the device buffer that the transaction will use. In most transaction types, information travels in both directions. The host initiates the transaction, data travels to or from the host, and the receiver of the data returns status information.

For high-speed traffic, the host divides each frame into eight microframes, each beginning with an SOF packet. Each microframe can carry multiple transactions to multiple destinations. Compared with full speed, individual transactions can carry more data, and protocol enhancements make better use of bus time at all speeds.

Hubs can increase the number of ports available to peripherals. A typical hub has one upstream port that transmits toward and receives from the host and as many as seven downstream ports that can connect to peripherals or additional hubs. A 1.x hub supports both low- and full-speed data rates but doesn't convert between speeds; it just passes the traffic on, changing only the edge rate to match the destination's speed. A 2.0 hub acts as a remote processor and converts from high-

speed data to low- or full-speed data as needed (**Figure 1**). The hubs' intelligence is a major reason for the high-speed bus's continued compatibility with 1.x hardware. It also means that 2.0 hubs are more internally complicated than 1.x hubs.

When you attach a device to a 1.x hub, the hub determines the device's speed by detecting the voltages on the D+ and D− signal wires from the device. If D− is pulled up at the device, its USB interface is low speed, and the hub passes only low-speed traffic to the device. In the other direction, the hub passes all the traffic that it receives from the low-speed device to the host. If D+ is pulled up at the device, the device's USB interface is full speed, and the hub passes low- and full-speed traffic in both directions. The hub passes low-speed traffic to full-speed devices because a full-speed device might be another hub with a low-speed device attached.

Because the high-speed bus is so much faster than low- and full-speed buses, high-speed hubs convert between speeds and use other tricks to keep slower data from clogging the bus. When you attach a 2.0 hub to a 2.0 host's root hub, all data between these hubs is high-speed data. To reduce jitter, the hub resynchronizes received high-speed data but otherwise passes it unchanged to any attached high-speed devices.

If you attach a low- or full-speed device to a hub that is receiving high-speed data from upstream, a transaction translator in the hub converts the data to low or full speed before passing it on. In the other direction, the hub converts low- or full-speed data to high speed before sending it toward the host.

TRANSFER TYPES: HIGH-SPEED DIFFERENCES

USB 2.0 supports the same four transfer types as 1.x. Control transfers are for enumeration and other times when the host wants to send defined requests and (optionally) receive data in reply. Interrupt transfers are for pointing devices and other applications that need to trans-

fer data at intervals, with a guaranteed maximum time between transactions. Bulk transfers are for printers, scanners, and other devices that would like to transfer data as quickly as possible but can wait if the bus is busy. Isochronous transfers are for real-time audio and other applications that require guaranteed delivery time but need no error correcting in the transfer.

A USB transaction consists of two or more packets. To begin a transaction, the host sends a token packet containing information about the transaction. A transaction must also have a data packet, a handshake packet where the receiver of the data returns status information, or both. (If no data packet exists, the device sends the status information.)

In addition to the high-speed bit rate, USB 2.0 has an improved protocol for high-speed transfers and new split transactions for full- and low-speed transfers on high-speed buses. **Table 1** compares USB's four transfer types at low, full, and high speeds. For bulk and isochronous transfers, high-speed transfers are about 40 times faster than full-speed transfers. No surprise there, because the bus speed is 40 times faster. But a high-speed interrupt transfer can be almost 400 times as much data per frame. Two reasons exist for this speed increase: The maximum packet size per transaction is much greater, and a transaction can have multiple data packets in a frame. High-speed control transfers are also much faster because they can transfer more data per frame.

For high-speed bulk and control transfers, 2.0 supports an improved protocol that uses less bus time to determine whether a device is ready to receive data. With full- and low-speed devices, when the host wants to send data in a control, bulk, or interrupt transfer, it sends the token and data packets and receives a reply from the device in the handshake packet of the transaction. If the device isn't ready for the data, it returns a NAK (negative acknowledge), and the host

TABLE 1—DATA-TRANSFER RATES TO OR FROM HIGH-SPEED ENDPOINTS			
	Maximum data-transfer rate/endpoint (kbytes/sec)		
Transfer type	**Low speed**	**Full speed**	**High speed**
Control	24	832	15,872
Interrupt	0.8	64	24,576
Bulk	Not allowed	1216	53,248
Isochronous	Not allowed	1023	24,576

tries again later. This protocol can waste a lot of bus time if the device is rarely ready.

High-speed bulk and control transactions have a better handshaking method. After receiving data, a device endpoint can return a NYET (not yet) handshake, which says that the endpoint accepted the data, but it is not yet ready to receive more data. When the host thinks the device might be ready, it sends a PING token packet, and the endpoint returns an ACK (acknowledge) or a NAK to indicate whether the host can send the next transaction's data.

To efficiently use bus time, high-speed hosts and hubs use new split transactions with low- and full-speed devices. At low and full speed, all of a transaction's packets are in sequence, with no other traffic between them. For example, on receiving token and data packets, a device must return an expected handshake packet without delay. But a high-speed hub could waste a lot of time waiting for a low- or full-speed device to receive the token and data packets and return a response.

The remedy is to do the transaction in two parts. At high speeds, the host sends the hub a start-split token packet along with any data the host is sending in the transaction. The host is then free to do other transactions without waiting for this transaction to complete. The high-speed hub then translates to low or full speed and completes the transaction with the destination device. However, instead of the hub passing the device's response to the host immediately upon receipt, it stores the response (which may be a data or handshake packet) in a buffer. Later, the host sends a complete-split token packet to request the device's response from the hub. The hub returns the data or handshake packet and completes the transaction with the host.

MORE FLEXIBLE AND RELIABLE TRANSFERS

Other improvements help to make USB more flexible and reliable. Interrupt and isochronous endpoints have a new requirement for a default low-bandwidth interface, and the intervals between transfers have more options. For isochronous transfers, the endpoint descriptor can now specify synchronization options.

A problem that surfaced with 1.x isochronous endpoints occurred when a device driver requested more bandwidth than was available on a busy bus. When the host configures a device that has an isochronous endpoint, it guarantees that the bandwidth will be available to transfer the requested amount of data in each frame or microframe. For example, on a full-speed bus, a device that wants to transfer 1023 bytes/frame requires 69% of the bus bandwidth. The host will thus refuse a request from another device to transfer 1023 bytes per frame. The user is left with a nonfunctioning device and may have no idea why.

To improve the situation, 2.0 has a new stipulation. The default interface for a device must have no isochronous payloads. In other words, any isochronous endpoint in the default interface must have a maximum packet size of zero.

This setup ensures that the host can configure the device no matter how busy the bus is. The device driver can then request an alternate interface that enables transferring isochronous data. The spec recommends providing several payload options to ensure that the device can make good use of the available bandwidth. A 1.x device can also provide these options, but 2.0 requires them.

Interrupt endpoints have a similar requirement. The data payload of the default interface can be no larger than 64 bytes. As with isochronous endpoints, the driver for a high-speed device can then request a larger payload. The default is the same as the full-speed maximum, so low- and full-speed endpoints need no change in payload size to comply with the spec.

Full- and high-speed 2.0 devices also have more flexibility in requesting the interval between transactions. USB 2.0 defines the bInterval field in the endpoint descriptor differently.

A full-speed isochronous endpoint can request a transaction every 1 to 32,768 msec (1.x's can do so only once every millisecond). For a high-speed endpoint, the range is 125 μsec to 4 sec.

For low- and full-speed interrupt endpoints, the range for maximum latency between transactions is unchanged (1 to 255 msec for full speed, 10 to 255 msec for low speed). A high-speed endpoint can have a maximum latency of 125 msec to 4 sec. Note that these times are maximums. The host may poll more often than the requested interval.

For isochronous endpoints, the descriptor can specify a synchronization and usage type. Synchronization describes the ability to time outgoing data in relation to incoming data. An example is a microphone that synchronizes to the USB's SOF signal to produce a fixed number of samples per microframe. Usage type specifies whether the endpoint carries data or feedback information.

UPDATING TO 2.0

So what does it take to update a 1.x device to 2.0? First of all, with the exception of hubs, 2.0 devices aren't required to support high speed. And 2.0 supports all of 1.x's device protocols.

The host identifies a 2.0-compliant device by the bcdUSB field in the device descriptor, which should be set to 0200h. Many devices need no other changes to comply with the new spec. Exceptions include some devices with isochronous endpoints and some low-speed devices. Remember, an isochronous endpoint must have a zero-bandwidth default interface.

If the device's interface is low speed, the cable must meet new shielding requirements. A 1.1-compliant low-speed cable requires no shielding. A 2.0-compliant low-speed cable must have the same aluminum-metallized-polyester inner shield and copper drain wire required for full- and high-speed cables. The spec also recommends, but doesn't require, a braided outer shield and a twisted pair for data, as on full- and high-speed cables.

The specification's authors did not add this change because the USB interface is electrically noisy, but did so because the

TABLE 2—OUTPUT VOLTAGES FOR LOW-SPEED, FULL-SPEED, AND HIGH-SPEED DRIVERS		
Driver voltage	Low/full speed (V)	High speed (V)
V_{OUT} low min	0	−0.01
V_{OUT} low max	0.3	0.01
V_{OUT} high min	2.8	0.36
V_{OUT} high max	3.6	0.44

High-speed devices must, of course, have controller chips that support the high-speed interface. Most support for the high-speed interface is in the controller's hardware, but the circuits between the chip's transceiver and the bus also require changes.

USB receivers detect a differential 0 or 1 by measuring whether the D+ or D− input is more positive. The voltage on each line must also be within a specified, absolute range. Transceivers must have separate drivers for high speed. For receiving, transceivers may have one pair of receivers that handles all speeds or separate receivers for high speeds. **Table 2** compares the driver-output voltages on low-, full-, and high-speed lines.

In a high-speed driver, a current source drives one line, with the other line at ground (**Figure 2**). To conserve power, a high-speed driver can activate its current source only when transmitting. This approach complicates the design, however, because the spec requires the device to meet amplitude and timing requirements from the very first symbol in a packet. So the spec also allows the driver to keep its current source active at all times, directing the current to ground when the device is not transmitting on the bus.

In a transceiver that is capable of high-speed data rates, the output impedance of the full-speed drivers has less tolerance (45Ω, $\pm10\%$ compared with 36Ω, $\pm22\%$). The change is necessary because the high-speed bus uses the full-speed drivers to terminate the line.

When the high-speed drivers are active, the full-speed drivers bring both data lines low (USB's single-ended zero state), resulting in each driver and its series resistor providing a 45Ω termination

acteristics. Receivers must include a differential envelope detector that detects the squelch (invalid signal) state.

Other new responsibilities for high-speed-capable controllers include managing the switch from full to high speed, detecting the removal of a high-speed device, and handling new protocols for entering and exiting the suspend and reset states.

In a low- or full-speed device, a 1.5-kΩ pullup resistor on one of the signal lines indicates device speed. Both wires also have 15-kΩ pulldown resistors at the hub. At high speeds, the pulldowns remain, but not the pullup. When a device switches to high speed, it must remove the pullup from the line.

When you attach a low- or full-speed device to the bus or remove one from the bus, the voltage change due to the pullup informs the hub of the change. High-speed devices always attach at full speed, so the hub detects these devices in the usual way.

The switch to high speed occurs during the reset signal, which the hub sends after it detects the device. A device that is capable of supporting high-speed data rates must support the new high-speed handshake that informs the hub that the device can handle high speeds and switches to high speed if possible.

The hub must also detect the removal of a high-speed device, which has no pullup. It does so by checking the voltage during the EOP (end-of-packet) signal in each high-speed SOF packet. When you remove a device from the bus, you remove its differential terminations, doubling the voltage at the hub's port. When the hub detects the doubled voltage, it knows that the device has been removed.

All USB devices must enter the low-

The standard control requests for devices and hubs have a few new options in 2.0. For example, a device may have configurations that are valid only at full or high speed. During enumeration, the host retrieves the descriptors that are valid for the current speed. Using the standard Get_Descriptor request with new values for descriptor type, the host can retrieve descriptors that are valid for the other speed.

High-speed devices must also support a new series of basic tests that the host selects with the Set_Feature request. These tests include setting the transceiver to defined states and sending test packets.

Many 1.x hubs use LEDs to indicate port status. The 2.0 spec has standard definitions for these displays. High-speed hubs have new requests to clear, reset, and stop the transaction translator buffer in the hub. There are also new hub-class requests to support the port indicators and to report whether a high-speed device is attached.□

AUTHOR BIOGRAPHY
Jan Axelson is the author of USB Complete: Everything You Need to Develop Custom USB Peripherals *(ISBN 0-9650819-3-1). She hosts a Web site for USB developers at www.lvr.com/usb.htm.*

REFERENCE
1. Wright, Maury, "USB and IEEE 1394: pretenders, contenders, or locks for ubiquitous desktop deployment?" *EDN*, April 25, 1996, http://www.ednmag.com/reg/1996/042596/09df1.htm#pic.